Addison Wesley Data & Analytics Series

R for Everyone

Advanced Analytics and Graphics

Jared P. Lander

R for Everyone

R for Everyone

Advanced Analytics and Graphics

Jared P. Lander

Addison-Wesley

Upper Saddle River, NJ • Boston • Indianapolis • San Francisco
New York • Toronto • Montreal • London • Munich • Paris • Madrid
Capetown • Sydney • Tokyo • Singapore • Mexico City

For information about buying this title in bulk quantities, or for special sales opportunities (which may include electronic versions; custom cover designs; and content particular to your business, training goals, marketing focus, or branding interests), please contact our corporate sales department at corpsales@pearsoned.com or (800) 382-3419.

For government sales inquiries, please contact governmentsales@pearsoned.com.

For questions about sales outside the U.S., please contact international@pearsoned.com.

Visit us on the Web: informit.com/aw

Library of Congress Cataloging-in-Publication Data

Lander, Jared P.
R for everyone / Jared P. Lander.
pages cm
Includes bibliographical references.
ISBN-13: 978-0-321-88803-7 (alk. paper)
ISBN-10: 0-321-88803-0 (alk. paper)
1. R (Computer program language) 2. Scripting languages (Computer science) 3. Statistics—Data processing. 4. Statistics—Graphic methods—Data processing. 5. Computer simulation. I. Title.
QA76.73.R3L36 2014
005.13–dc23
2013027407

ISBN-13: 978-0-321-88803-7
ISBN-10: 0-321-88803-0
Text printed in the United States on recycled paper at RR Donnelley in Crawfordsville, Indiana.
First printing, December 2013

❖

To my mother and father

❖

This page intentionally left blank

Contents

Foreword

R has had tremendous growth in popularity over the last three years. Based on that, you'd think that it was a new, up-and-coming language. But surprisingly, R has been around since 1993. Why the sudden uptick in popularity? The somewhat obvious answer seems to be the emergence of data science as a career and a field of study. But the underpinnings of data science have been around for many decades. Statistics, linear algebra, operations research, artificial intelligence, and machine learning all contribute parts to the tools that a modern data scientist uses. R, more than most languages, has been built to make most of these tools only a single function call away.

That's why I'm very excited to have this book as one of the first in the Addison-Wesley Data and Analytics Series. R is indispensable for many data science tasks. Many algorithms useful for prediction and analysis can be accessed through only a few lines of code, which makes it a great fit for solving modern data challenges. Data science as a field isn't just about math and statistics, and it isn't just about programming and infrastructure. This book provides a well-balanced introduction to the power and expressiveness of R and is aimed at a general audience.

I can't think of a better author to provide an introduction to R than Jared Lander. Jared and I first met through the New York City machine learning community in late 2009. Back then, the New York City data community was small enough to fit in a single conference room, and many of the other data meetups had yet to be formed. Over the last four years, Jared has been at the forefront of the emerging data science profession.

Through running the Open Statistical Programming Meetup, speaking at events, and teaching a course at Columbia on R, Jared has helped grow the community by educating programmers, data scientists, journalists, and statisticians alike. But Jared's expertise isn't limited to teaching. As an everyday practitioner, he puts these tools to use while consulting for clients big and small.

This book provides an introduction both to programming in R and to the various statistical methods and tools an everyday R programmer uses. Examples use publicly available datasets that Jared has helpfully cleaned and made accessible through his Web site. By using real data and setting up interesting problems, this book stays engaging to the end.

—*Paul Dix, Series Editor*

This page intentionally left blank

Preface

With the increasing prevalence of data in our daily lives, new and better tools are needed to analyze the deluge. Traditionally there have been two ends of the spectrum: lightweight, individual analysis using tools like Excel or SPSS and heavy duty, high-performance analysis built with C++ and the like. With the increasing strength of personal computers grew a middle ground that was both interactive and robust. Analysis done by an individual on his or her own computer in an exploratory fashion could quickly be transformed into something destined for a server, underpinning advanced business processes. This area is the domain of R, Python, and other scripted languages.

R, invented by Robert Gentleman and Ross Ihaka of the University of Auckland in 1993, grew out of S, which was invented by John Chambers at Bell Labs. It is a high-level language that was originally intended to be run interactively where the user runs a command, gets a result, and then runs another command. It has since evolved into a language that can also be embedded in systems and tackle complex problems.

In addition to transforming and analyzing data, R can produce amazing graphics and reports with ease. It is now being used as a full stack for data analysis, extracting and transforming data, fitting models, drawing inferences and making predictions, plotting and reporting results.

R's popularity has skyrocketed since the late 2000s, as it has stepped out of academia and into banking, marketing, pharmaceuticals, politics, genomics and many other fields. Its new users are often shifting from low-level, compiled languages like C++, other statistical packages such as SAS or SPSS, and from the 800-pound gorilla, Excel. This time period also saw a rapid surge in the number of add-on packages—libraries of prewritten code that extend R's functionality.

While R can sometimes be intimidating to beginners, especially for those without programming experience, I find that programming analysis, instead of pointing and clicking, soon becomes much easier, more convenient and more reliable. It is my goal to make that learning process easier and quicker.

This book lays out information in a way I wish I were taught when learning R in graduate school. Coming full circle, the content of this book was developed in conjuction with the data science course I teach at Columbia University. It is not meant to cover every minute detail of R, but rather the 20% of functionality needed to accomplish 80% of the work. The content is organized into self-contained chapters as follows.

Chapter 1, Getting R: Where to download R and how to install it. This deals with the varying operating systems and 32-bit versus 64-bit versions. It also gives advice on where to install R.

Chapter 2, The `R` Environment: An overview of using `R`, particularly from within RStudio. RStudio projects and Git integration are covered as is customizing and navigating RStudio.

Chapter 3, Packages: How to locate, install and load `R` packages.

Chapter 4, Basics of `R`: Using `R` for math. Variable types such as `numeric`, `character` and `Date` are detailed as are `vectors`. There is a brief introduction to calling functions and finding documentation on functions.

Chapter 5, Advanced Data Structures: The most powerful and commonly used data structure, `data.frames`, along with `matrices` and `lists`, are introduced.

Chapter 6, Reading Data into `R`: Before data can be analyzed it must be read into `R`. There are numerous ways to ingest data, including reading from CSVs and databases.

Chapter 7, Statistical Graphics: Graphics are a crucial part of preliminary data analysis and communicating results. `R` can make beautiful plots using its powerful plotting utilities. Base graphics and `ggplot2` are introduced and detailed here.

Chapter 8, Writing `R` Functions: Repeatable analysis is often made easier with user-defined functions. The structure, arguments and return rules are discussed.

Chapter 9, Control Statements: Controlling the flow of programs using `if`, `ifelse` and complex checks.

Chapter 10, Loops, the Un-`R` Way to Iterate: Iterating using `for` and `while` loops. While these are generally discouraged they are important to know.

Chapter 11, Group Manipulation: A better alternative to loops, vectorization does not quite iterate through data so much as operate on all elements at once. This is more efficient and is primarily performed with the `apply` functions and `plyr` package.

Chapter 12, Data Reshaping: Combining multiple datasets, whether by stacking or joining, is commonly necessary as is changing the shape of data. The `plyr` and `reshape2` packages offer good functions for accomplishing this in addition to base tools such as `rbind`, `cbind` and `merge`.

Chapter 13, Manipulating Strings: Most people do not associate character data with statistics but it is an important form of data. `R` provides numerous facilities for working with strings, including combining them and extracting information from within. Regular expressions are also detailed.

Chapter 14, Probability Distributions: A thorough look at the normal, binomial and Poisson distributions. The formulas and functions for many distributions are noted.

Chapter 15, Basic Statistics: These are the first statistics most people are taught, such as mean, standard deviation and t-tests.

Chapter 16, Linear Models: The most powerful and common tool in statistics, linear models are extensively detailed.

Chapter 17, Generalized Linear Models: Linear models are extended to include logistic and Poisson regression. Survival analysis is also covered.

Chapter 18, Model Diagnostics: Determining the quality of models and variable selection using residuals, AIC, cross-validation, the bootstrap and stepwise variable selection.

Chapter 19, Regularization and Shrinkage: Preventing overfitting using the Elastic Net and Bayesian methods.

Chapter 20, Nonlinear Models: When linear models are inappropriate, nonlinear models are a good solution. Nonlinear least squares, splines, generalized additive models, decision trees and random forests are discussed.

Chapter 21, Time Series and Autocorrelation: Methods for the analysis of univariate and multivariate time series data.

Chapter 22, Clustering: Clustering, the grouping of data, is accomplished by various methods such as K-means and hierarchical clustering.

Chapter 23, Reproducibility, Reports and Slide Shows with `knitr`: Generating reports, slide shows and Web pages from within `R` is made easy with `knitr`, LaTeX and Markdown.

Chapter 24, Building `R` Packages: `R` packages are great for portable, reusable code. Building these packages has been made incredibly easy with the advent of `devtools` and `Rcpp`.

Appendix A, Real-Life Resources: A listing of our favorite resources for learning more about `R` and interacting with the community.

Appendix B, Glossary: A glossary of terms used throughout this book.

A good deal of the text in this book is either `R` code or the results of running code. Code and results are most often in a separate block of text and set in a distinctive font, as shown in the following example. The different parts of code also have different colors. Lines of code start with $>$, and if code is continued from one line to another the continued line begins with $+$.

```
> # this is a comment
>
> # now basic math
> 10 * 10

[1] 100

>
> # calling a function
> sqrt(4)

[1] 2
```

Certain Kindle devices do not display color so the digital edition of this book will be viewed in greyscale on those devices.

There are occasions where code is shown inline and looks like `sqrt(4)`.

In the few places where math is necessary, the equations are indented from the margin and are numbered.

$$e^{i\pi} + 1 = 0 \tag{1}$$

Within equations, normal variables appear as italic text (x), vectors are bold lowercase letters ($\mathbf{x}$) and matrices are bold uppercase letters ($\mathbf{X}$). Greek letters, such as α and β, follow the same convention.

Function names will be written as `join` and package names as `plyr`. Objects generated in code that are referenced in text are written as `object1`.

Learning `R` is a gratifying experience that makes life so much easier for so many tasks. I hope you enjoy learning with me.

Acknowledgments

To start, I must thank my mother, Gail Lander, for encouraging me to become a math major. Without that I would never have followed the path that led me to statistics and data science. In a similar vein, I have to thank my father, Howard Lander, for paying all those tuition bills. He has been a valuable source of advice and guidance throughout my life and someone I have aspired to emulate in many ways. While they both insist they do not understand what I do, they love that I do it and have helped me all along the way. Staying with family, I should thank my sister and brother-in-law, Aimee and Eric Schechterman, for letting me teach math to Noah, their five-year-old son.

There are many teachers who have helped shape me over the years. The first is Rochelle Lecke, who tutored me in middle school math even when my teacher told me I did not have worthwhile math skills.

Then there is Beth Edmondson, my precalc teacher at Princeton Day School. After I wasted the first half of high school as a mediocre student, she told me I had "some nerve signing up for next year's AP Calc" given my grades. She agreed to let me take AP Calc if I went from a C to an A+ in her class, never thinking I stood a chance. Three months later, she was in shock as I not only earned the A+, but turned around my entire academic career. She changed my life and without her, I do not know where I would be today. I am forever grateful that she was my teacher.

For the first two years at Muhlenberg College, I was determined to be a business and communications major, but took math classes because they came naturally to me. My professors, Dr. Penny Dunham, Dr. Bill Dunham, and Dr. Linda McGuire, all convinced me to become a math major, a decision that has greatly shaped my life. Dr. Greg Cicconetti gave me my first glimpse of rigorous statistics, my first research opportunity and planted the idea in my head that I should go to grad school for statistics.

While earning my M.A. at Columbia University, I was surrounded by brilliant minds in statistics and programming. Dr. David Madigan opened my eyes to modern machine learning, and Dr. Bodhi Sen got me thinking about statistical programming. I had the privilege to do research with Dr. Andrew Gelman, whose insights have been immeasurably important to me. Dr. Richard Garfield showed me how to use statistics to help people in disaster and war zones when he sent me on my first assignment to Myanmar. His advice and friendship over the years have been dear to me. Dr. Jingchen Liu allowed and encouraged me to write my thesis on New York City pizza, which has brought me an inordinate amount of attention.[1]

While at Columbia, I also met my good friend—and one time TA— Dr. Ivor Cribben who filled in so many gaps in my knowledge. Through him, I met Dr. Rachel Schutt, a source of great advice, and who I am now honored to teach alongside at Columbia.

Grad school might never have happened without the encouragement and support of Shanna Lee. She helped maintain my sanity while I was incredibly overcommited to two jobs, classes and Columbia's hockey team. I am not sure I would have made it through without her.

Steve Czetty gave me my first job in analytics at Sky IT Group and taught me about databases, while letting me experiment with off-the-wall programming. This sparked my interest in statistics and data. Joe DeSiena, Philip du Plessis, and Ed Bobrin at the Bardess Group are some of the finest people I have ever had the pleasure to work with, and I am proud to be working with them to this day. Mike Minelli, Rich Kittler, Mark Barry, David Smith, Joseph Rickert, Dr. Norman Nie, James Peruvankal, Neera Talbert and Dave Rich at Revolution Analytics let me do one of the best jobs I could possibly imagine: explaining to people in business why they should be using `R`. Kirk Mettler, Richard Schultz, Dr. Bryan Lewis and Jim Winfield at Big Computing encouraged me to have fun, tackling interesting problems in `R`. Vincent Saulys, John Weir, and Dr. Saar Golde at Goldman Sachs made my time there both enjoyable and educational.

Throughout the course of writing this book, many people helped me with the process. First and foremost is Yin Cheung, who saw all the stress I constantly felt and supported me through many ruined nights and days.

My editor, Debra Williams, knew just how to encourage me and her guiding hand has been invaluable. Paul Dix, the series editor and a good friend, was the person who suggested I write this book, so none of this would have happened without him. Thanks to Caroline Senay and Andrea Fox for being great copy editors. Without them, this book would not be nearly as well put together. Robert Mauriello's technical review was incredibly useful in honing the book's presentation.

The folks at RStudio, particularly JJ Allaire and Josh Paulson, make an amazing product, which made the writing process far easier than it would have been otherwise. Yihui Xie, the author of the `knitr` package, provided numerous feature changes that I needed to write this book. His software, and his speed at implementing my requests, is greatly appreciated.

Numerous people have provided valuable feedback as I produced this book, including Chris Bethel, Dr. Dirk Eddelbuettel, Dr. Ramnath Vaidyanathan, Dr. Eran Bellin,

1. `http://slice.seriouseats.com/archives/2010/03/the-moneyball-of-pizza-statistician-uses-statistics-to-find-nyc-best-pizza.html`

Avi Fisher, Brian Ezra, Paul Puglia, Nicholas Galasinao, Aaron Schumaker, Adam Hogan, Jeffrey Arnold, and John Houston.

Last fall was my first time teaching, and I am thankful to the students from the Fall 2012 Introduction to Data Science class at Columbia University for being the guinea pigs for the material that ultimately ended up in this book.

Thank you to everyone who helped along the way.

This page intentionally left blank

About the Author

Jared P. Lander is the founder and CEO of Lander Analytics, a statistical consulting firm based in New York City, the organizer of the New York Open Statistical Programming Meetup, and an adjunct professor of statistics at Columbia University. He is also a tour guide for Scott's Pizza Tours and an advisor to Brewla Bars, a gourmet ice pop start-up. With an M.A. from Columbia University in statistics and a B.A. from Muhlenberg College in mathematics, he has experience in both academic research and industry. His work for both large and small organizations spans politics, tech start-ups, fund-raising, music, finance, healthcare and humanitarian relief efforts.

He specializes in data management, multilevel models, machine learning, generalized linear models, visualization, data management and statistical computing.

This page intentionally left blank

This page intentionally left blank

Chapter 12

Data Reshaping

As noted in Chapter 11, manipulating the data takes a great deal of effort before serious analysis can begin. In this chapter we will consider when the data needs to be rearranged from column oriented to row oriented (or the opposite) and when the data are in multiple, separate sets and need to be combined into one.

There are base functions to accomplish these tasks but we will focus on those in `plyr`, `reshape2` and `data.table`.

12.1 `cbind` and `rbind`

The simplest case is when we have two datasets with either identical columns (both the number of and names) or the same number of rows. In this case, either `rbind` or `cbind` work great.

As a first trivial example, we create two simple `data.frame`s by combining a few vectors with `cbind`, and then stack them using `rbind`.

```
> # make two vectors and combine them as columns in a data.frame
> sport <- c("Hockey", "Baseball", "Football")
> league <- c("NHL", "MLB", "NFL")
> trophy <- c("Stanley Cup", "Commissioner's Trophy",
+             "Vince Lombardi Trophy")
> trophies1 <- cbind(sport, league, trophy)
> # make another data.frame using data.frame()
> trophies2 <- data.frame(sport=c("Basketball", "Golf"),
+                         league=c("NBA", "PGA"),
+                         trophy=c("Larry O'Brien Championship Trophy",
+                                  "Wanamaker Trophy"),
+                         stringsAsFactors=FALSE)
> # combine them into one data.frame with rbind
> trophies <- rbind(trophies1, trophies2)
```

Both cbind and rbind can take multiple arguments to combine an arbitrary number of objects. Note that it is possible to assign new column names to vectors in cbind.

```
> cbind(Sport = sport, Association = league, Prize = trophy)

     Sport      Association Prize
[1,] "Hockey"   "NHL"       "Stanley Cup"
[2,] "Baseball" "MLB"       "Commissioner's Trophy"
[3,] "Football" "NFL"       "Vince Lombardi Trophy"
```

12.2 Joins

Data do not always come so nicely aligned for combining using cbind, so they need to be joined together using a common key. This concept should be familiar to SQL users. Joins in R are not as flexible as SQL joins, but are still an essential operation in the data analysis process.

The three most commonly used functions for joins are merge in base R, join in plyr and the merging functionality in data.table. Each has pros and cons with some pros outweighing their respective cons.

To illustrate these functions I have prepared data originally made available as part of the USAID Open Government initiative.[1] The data have been chopped into eight separate files so that they can be joined together. They are all available in a zip file at http://jaredlander.com/data/US_Foreign_Aid.zip. These should be downloaded and unzipped to a folder on our computer. This can be done a number of ways (including using a mouse!) but we show how to download and unzip using R.

```
> download.file(url="http://jaredlander.com/data/US_Foreign_Aid.zip",
+               destfile="data/ForeignAid.zip")
> unzip("data/ForeignAid.zip", exdir="data")
```

To load all of these files programmatically, we use a for loop as seen in Section 10.1. We get a list of the files using dir, and then loop through that list assigning each dataset to a name specified using assign.

```
> require(stringr)
> # first get a list of the files
> theFiles <- dir("data/", pattern="\\.csv")
> ## loop through those files
> for(a in theFiles)
+ {
+     # build a good name to assign to the data
+     nameToUse <- str_sub(string=a, start=12, end=18)
```

1. More information about the data is available at http://gbk.eads.usaidallnet.gov/.

```
+     # read in the csv using read.table
+     # file.path is a convenient way to specify a folder and file name
+     temp <- read.table(file=file.path("data", a),
+                        header=TRUE, sep=",", stringsAsFactors=FALSE)
+     # assign them into the workspace
+     assign(x=nameToUse, value=temp)
+ }
```

12.2.1 merge

R comes with a built-in function, called `merge`, to merge two `data.frames`.

```
> Aid90s00s <- merge(x=Aid_90s, y=Aid_00s,
+                    by.x=c("Country.Name", "Program.Name"),
+                    by.y=c("Country.Name", "Program.Name"))
> head(Aid90s00s)

  Country.Name                                        Program.Name
1  Afghanistan                          Child Survival and Health
2  Afghanistan          Department of Defense Security Assistance
3  Afghanistan                              Development Assistance
4  Afghanistan Economic Support Fund/Security Support Assistance
5  Afghanistan                                  Food For Education
6  Afghanistan                    Global Health and Child Survival
  FY1990 FY1991 FY1992   FY1993  FY1994 FY1995 FY1996 FY1997 FY1998
1     NA     NA     NA       NA      NA     NA     NA     NA     NA
2     NA     NA     NA       NA      NA     NA     NA     NA     NA
3     NA     NA     NA       NA      NA     NA     NA     NA     NA
4     NA     NA     NA 14178135 2769948     NA     NA     NA     NA
5     NA     NA     NA       NA      NA     NA     NA     NA     NA
6     NA     NA     NA       NA      NA     NA     NA     NA     NA
  FY1999 FY2000  FY2001   FY2002    FY2003           FY2004     FY2005
1     NA     NA      NA  2586555  56501189         40215304   39817970
2     NA     NA      NA  2964313        NA         45635526  151334908
3     NA     NA 4110478  8762080  54538965        180539337  193598227
4     NA     NA   61144 31827014 341306822       1025522037 1157530168
5     NA     NA      NA       NA   3957312          2610006    3254408
6     NA     NA      NA       NA        NA               NA         NA
      FY2006     FY2007     FY2008     FY2009
1   40856382   72527069   28397435         NA
2  230501318  214505892  495539084  552524990
3  212648440  173134034  150529862    3675202
4 1357750249 1266653993 1400237791 1418688520
5     386891         NA         NA         NA
6         NA         NA   63064912    1764252
```

The `by.x` specifies the key column(s) in the left `data.frame` and `by.y` does the same for the right `data.frame`. The ability to specify different column names for each `data.frame` is the most useful feature of `merge`. The biggest drawback, however, is that `merge` can be much slower than the alternatives.

12.2.2 plyr join

Returning to Hadley Wickham's `plyr` package, we see it includes a `join` function, which works similarly to `merge` but is much faster. The biggest drawback, though, is that the key column(s) in each table must have the same name. We use the same data used previously to illustrate.

```
> require(plyr)
> Aid90s00sJoin <- join(x = Aid_90s, y = Aid_00s, by = c("Country.Name",
+     "Program.Name"))
> head(Aid90s00sJoin)

  Country.Name                                      Program.Name
1  Afghanistan                         Child Survival and Health
2  Afghanistan         Department of Defense Security Assistance
3  Afghanistan                            Development Assistance
4  Afghanistan Economic Support Fund/Security Support Assistance
5  Afghanistan                                Food For Education
6  Afghanistan                  Global Health and Child Survival
  FY1990 FY1991 FY1992   FY1993  FY1994 FY1995 FY1996 FY1997 FY1998
1     NA     NA     NA       NA      NA     NA     NA     NA     NA
2     NA     NA     NA       NA      NA     NA     NA     NA     NA
3     NA     NA     NA       NA      NA     NA     NA     NA     NA
4     NA     NA     NA 14178135 2769948     NA     NA     NA     NA
5     NA     NA     NA       NA      NA     NA     NA     NA     NA
6     NA     NA     NA       NA      NA     NA     NA     NA     NA
  FY1999 FY2000  FY2001   FY2002    FY2003     FY2004     FY2005
1     NA     NA      NA  2586555  56501189   40215304   39817970
2     NA     NA      NA  2964313        NA   45635526  151334908
3     NA     NA 4110478  8762080  54538965  180539337  193598227
4     NA     NA   61144 31827014 341306822 1025522037 1157530168
5     NA     NA      NA       NA   3957312    2610006    3254408
6     NA     NA      NA       NA        NA         NA         NA
      FY2006     FY2007     FY2008     FY2009
1   40856382   72527069   28397435         NA
2  230501318  214505892  495539084  552524990
3  212648440  173134034  150529862    3675202
4 1357750249 1266653993 1400237791 1418688520
5     386891         NA         NA         NA
6         NA         NA   63064912    1764252
```

`join` has an argument for specifying a left, right, inner or full (outer) join.

We have eight data.frames containing foreign assistance data that we would like to combine into one data.frame without hand coding each join. The best way to do this is to put all the data.frames into a list, and then successively join them together using Reduce.

```
> # first figure out the names of the data.frames
> frameNames <- str_sub(string = theFiles, start = 12, end = 18)
> # build an empty list
> frameList <- vector("list", length(frameNames))
> names(frameList) <- frameNames
> # add each data.frame into the list
> for (a in frameNames)
+ {
+     frameList[[a]] <- eval(parse(text = a))
+ }
```

A lot happened in that section of code, so let's go over it carefully. First we reconstructed the names of the data.frames using str_sub from Hadley Wickham's stringr package, which is shown in more detail in Chapter 13. Then we built an empty list with as many elements as there are data.frames, in this case eight, using vector and assigning its mode to "list." We then set appropriate names to the list.

Now that the list is built and named, we loop through it, assigning to each element the appropriate data.frame. The problem is that we have the names of the data.frames as characters but the <- operator requires a variable, not a character. So we parse and evaluate the character, which realizes the actual variable. Inspecting, we see that the list does indeed contain the appropriate data.frames.

```
> head(frameList[[1]])

  Country.Name                                     Program.Name
1  Afghanistan                        Child Survival and Health
2  Afghanistan        Department of Defense Security Assistance
3  Afghanistan                           Development Assistance
4  Afghanistan Economic Support Fund/Security Support Assistance
5  Afghanistan                               Food For Education
6  Afghanistan                 Global Health and Child Survival
  FY2000  FY2001   FY2002    FY2003     FY2004     FY2005     FY2006
1     NA      NA  2586555  56501189   40215304   39817970   40856382
2     NA      NA  2964313        NA   45635526   45635526  230501318
3     NA 4110478  8762080  54538965  180539337  193598227  212648440
4     NA   61144 31827014 341306822 1025522037 1157530168 1357750249
5     NA      NA       NA   3957312    2610006    3254408     386891
6     NA      NA       NA        NA         NA         NA         NA
```

```
     FY2007     FY2008     FY2009
1   72527069   28397435         NA
2  214505892  495539084  552524990
3  173134034  150529862    3675202
4 1266653993 1400237791 1418688520
5         NA         NA         NA
6         NA   63064912    1764252

> head(frameList[["Aid_00s"]])

  Country.Name                                   Program.Name
1  Afghanistan                        Child Survival and Health
2  Afghanistan          Department of Defense Security Assistance
3  Afghanistan                           Development Assistance
4  Afghanistan Economic Support Fund/Security Support Assistance
5  Afghanistan                               Food For Education
6  Afghanistan                  Global Health and Child Survival
  FY2000  FY2001   FY2002    FY2003     FY2004     FY2005     FY2006
1     NA      NA  2586555  56501189   40215304   39817970   40856382
2     NA      NA  2964313        NA   45635526  151334908  230501318
3     NA 4110478  8762080  54538965  180539337  193598227  212648440
4     NA   61144 31827014 341306822 1025522037 1157530168 1357750249
5     NA      NA       NA   3957312    2610006    3254408     386891
6     NA      NA       NA        NA         NA         NA         NA
     FY2007     FY2008     FY2009
1   72527069   28397435         NA
2  214505892  495539084  552524990
3  173134034  150529862    3675202
4 1266653993 1400237791 1418688520
5         NA         NA         NA
6         NA   63064912    1764252

> head(frameList[[5]])

  Country.Name                                   Program.Name
1  Afghanistan                        Child Survival and Health
2  Afghanistan          Department of Defense Security Assistance
3  Afghanistan                           Development Assistance
4  Afghanistan Economic Support Fund/Security Support Assistance
5  Afghanistan                               Food For Education
6  Afghanistan                  Global Health and Child Survival
  FY1960 FY1961    FY1962 FY1963 FY1964 FY1965 FY1966 FY1967 FY1968
1     NA     NA        NA     NA     NA     NA     NA     NA     NA
2     NA     NA        NA     NA     NA     NA     NA     NA     NA
3     NA     NA        NA     NA     NA     NA     NA     NA     NA
```

```
4      NA      NA 181177853      NA      NA      NA      NA      NA      NA
5      NA      NA        NA      NA      NA      NA      NA      NA      NA
6      NA      NA        NA      NA      NA      NA      NA      NA      NA
  FY1969
1      NA
2      NA
3      NA
4      NA
5      NA
6      NA
```

```
> head(frameList[["Aid_60s"]])
```

```
  Country.Name                                              Program.Name
1  Afghanistan                                 Child Survival and Health
2  Afghanistan         Department of Defense Security Assistance
3  Afghanistan                                    Development Assistance
4  Afghanistan Economic Support Fund/Security Support Assistance
5  Afghanistan                                        Food For Education
6  Afghanistan                            Global Health and Child Survival
  FY1960 FY1961    FY1962 FY1963 FY1964 FY1965 FY1966 FY1967 FY1968
1      NA      NA        NA      NA      NA      NA      NA      NA      NA
2      NA      NA        NA      NA      NA      NA      NA      NA      NA
3      NA      NA        NA      NA      NA      NA      NA      NA      NA
4      NA      NA 181177853      NA      NA      NA      NA      NA      NA
5      NA      NA        NA      NA      NA      NA      NA      NA      NA
6      NA      NA        NA      NA      NA      NA      NA      NA      NA
  FY1969
1      NA
2      NA
3      NA
4      NA
5      NA
6      NA
```

Having all the `data.frames` in a `list` allows us to iterate through the `list`, joining all the elements together (or applying any function to the elements iteratively). Rather than using a loop, we use the `Reduce` function to speed up the operation.

```
> allAid <- Reduce(function(...)
+ {
+     join(..., by = c("Country.Name", "Program.Name"))
+ }, frameList)
> dim(allAid)
```

```
[1] 2453   67
```

```
> require(useful)
> corner(allAid, c = 15)

  Country.Name                                      Program.Name
1  Afghanistan                        Child Survival and Health
2  Afghanistan         Department of Defense Security Assistance
3  Afghanistan                           Development Assistance
4  Afghanistan Economic Support Fund/Security Support Assistance
5  Afghanistan                               Food For Education
  FY2000  FY2001   FY2002    FY2003     FY2004     FY2005     FY2006
1     NA      NA  2586555  56501189   40215304   39817970   40856382
2     NA      NA  2964313        NA   45635526  151334908  230501318
3     NA 4110478  8762080  54538965  180539337  193598227  212648440
4     NA   61144 31827014 341306822 1025522037 1157530168 1357750249
5     NA      NA       NA   3957312    2610006    3254408     386891
      FY2007     FY2008     FY2009     FY2010  FY1946 FY1947
1   72527069   28397435         NA         NA      NA     NA
2  214505892  495539084  552524990  316514796      NA     NA
3  173134034  150529862    3675202         NA      NA     NA
4 1266653993 1400237791 1418688520 2797488331      NA     NA
5         NA         NA         NA         NA      NA     NA

> bottomleft(allAid, c = 15)

     Country.Name           Program.Name  FY2000  FY2001   FY2002
2449     Zimbabwe Other State Assistance 1341952  322842       NA
2450     Zimbabwe Other USAID Assistance 3033599 8464897  6624408
2451     Zimbabwe            Peace Corps 2140530 1150732   407834
2452     Zimbabwe                Title I      NA      NA       NA
2453     Zimbabwe               Title II      NA      NA 31019776
       FY2003   FY2004   FY2005  FY2006    FY2007    FY2008    FY2009
2449       NA   318655    44553  883546   1164632   2455592   2193057
2450 11580999 12805688 10091759 4567577  10627613  11466426  41940500
2451       NA       NA       NA      NA        NA        NA        NA
2452       NA       NA       NA      NA        NA        NA        NA
2453       NA       NA       NA  277468 100053600 180000717 174572685
       FY2010 FY1946 FY1947
2449  1605765     NA     NA
2450 30011970     NA     NA
2451       NA     NA     NA
2452       NA     NA     NA
2453 79545100     NA     NA
```

`Reduce` can be a difficult function to grasp, so we illustrate it with a simple example. Let's say we have a `vector` of the first ten integers, `1:10`, and want to sum them (forget for a moment that `sum(1:10)` will work perfectly). We can call `Reduce(sum, 1:10)`, which will first add 1 and 2. It will then add 3 to that result, then 4 to that result, and so on, resulting in 55.

Likewise, we passed a `list` to a function that joins its inputs, which in this case was simply `...`, meaning that anything could be passed. Using `...` is an advanced trick of `R` programming that can be difficult to get right. `Reduce` passed the first two `data.frames` in the `list`, which were then joined. That result was then joined to the next `data.frame` and so on until they were all joined together.

12.2.3 `data.table` merge

Like many other operations in `data.table`, joining data requires a different syntax, and possibly a different way of thinking. To start, we convert two of our foreign aid datasets' `data.frames` into `data.tables`.

```
> require(data.table)
> dt90 <- data.table(Aid_90s, key = c("Country.Name", "Program.Name"))
> dt00 <- data.table(Aid_00s, key = c("Country.Name", "Program.Name"))
```

Then, doing the join is a simple operation. Note that the join requires specifying the keys for the `data.tables`, which we did during their creation.

```
> dt0090 <- dt90[dt00]
```

In this case `dt90` is the left side, `dt00` is the right side and a left join was performed.

12.3 `reshape2`

The next most common munging need is either melting data (going from column orientation to row orientation) or casting data (going from row orientation to column orientation). As with most other procedures in `R`, there are multiple functions available to accomplish these tasks but we will focus on Hadley Wickham's `reshape2` package. (We talk about Wickham a lot because his products have become so fundamental to the `R` developer's toolbox.)

12.3.1 `melt`

Looking at the `Aid_00s` `data.frame`, we see that each year is stored in its own column. That is, the dollar amount for a given country and program is found in a different column for each year. This is called a cross table, which, while nice for human consumption, is not ideal for graphing with `ggplot2` or for some analysis algorithms.

```
> head(Aid_00s)

  Country.Name                                   Program.Name
1  Afghanistan                      Child Survival and Health
2  Afghanistan         Department of Defense Security Assistance
```

```
3  Afghanistan                             Development Assistance
4  Afghanistan Economic Support Fund/Security Support Assistance
5  Afghanistan                                   Food For Education
6  Afghanistan                       Global Health and Child Survival
  FY2000  FY2001   FY2002    FY2003     FY2004     FY2005     FY2006
1     NA      NA  2586555  56501189   40215304   39817970   40856382
2     NA      NA  2964313        NA   45635526  151334908  230501318
3     NA 4110478  8762080  54538965  180539337  193598227  212648440
4     NA   61144 31827014 341306822 1025522037 1157530168 1357750249
5     NA      NA       NA   3957312    2610006    3254408     386891
6     NA      NA       NA        NA         NA         NA         NA
      FY2007     FY2008     FY2009
1   72527069   28397435         NA
2  214505892  495539084  552524990
3  173134034  150529862    3675202
4 1266653993 1400237791 1418688520
5         NA         NA         NA
6         NA   63064912    1764252
```

We want it set up so that each row represents a single country-program-year entry with the dollar amount stored in one column. To achieve this we melt the data using `melt` from `reshape2`.

```
> require(reshape2)
> melt00 <- melt(Aid_00s, id.vars=c("Country.Name", "Program.Name"),
+                variable.name="Year", value.name="Dollars")
> tail(melt00, 10)

      Country.Name
24521     Zimbabwe
24522     Zimbabwe
24523     Zimbabwe
24524     Zimbabwe
24525     Zimbabwe
24526     Zimbabwe
24527     Zimbabwe
24528     Zimbabwe
24529     Zimbabwe
24530     Zimbabwe
                                    Program.Name   Year
24521           Migration and Refugee Assistance FY2009
24522                         Narcotics Control FY2009
```

```
24523 Nonproliferation, Anti-Terrorism, Demining and Related FY2009
24524                             Other Active Grant Programs FY2009
24525                                 Other Food Aid Programs FY2009
24526                                  Other State Assistance FY2009
24527                                  Other USAID Assistance FY2009
24528                                             Peace Corps FY2009
24529                                                 Title I FY2009
24530                                                Title II FY2009
        Dollars
24521   3627384
24522        NA
24523        NA
24524   7951032
24525        NA
24526   2193057
24527  41940500
24528        NA
24529        NA
24530 174572685
```

The `id.vars` argument specifies which columns uniquely identify a row.

After some manipulation of the `Year` column and aggregating, this is now prime for plotting, as shown in Figure 12.1. The plot uses faceting allowing us to quickly see and understand the funding for each program over time.

```
> require(scales)
> # strip the "FY" out of the year column and convert it to numeric
> melt00$Year <- as.numeric(str_sub(melt00$Year, start=3, 6))
> # aggregate the data so we have yearly numbers by program
> meltAgg <- aggregate(Dollars ~ Program.Name + Year, data=melt00,
+                      sum, na.rm=TRUE)
> # just keep the first 10 characters of program name
> # then it will fit in the plot
> meltAgg$Program.Name <- str_sub(meltAgg$Program.Name, start=1,
+                                 end=10)
>
> ggplot(meltAgg, aes(x=Year, y=Dollars)) +
+     geom_line(aes(group=Program.Name)) +
+     facet_wrap(~ Program.Name) +
+     scale_x_continuous(breaks=seq(from=2000, to=2009, by=2)) +
+     theme(axis.text.x=element_text(angle=90, vjust=1, hjust=0)) +
+     scale_y_continuous(labels=multiple_format(extra=dollar,
+                                               multiple="B"))
```

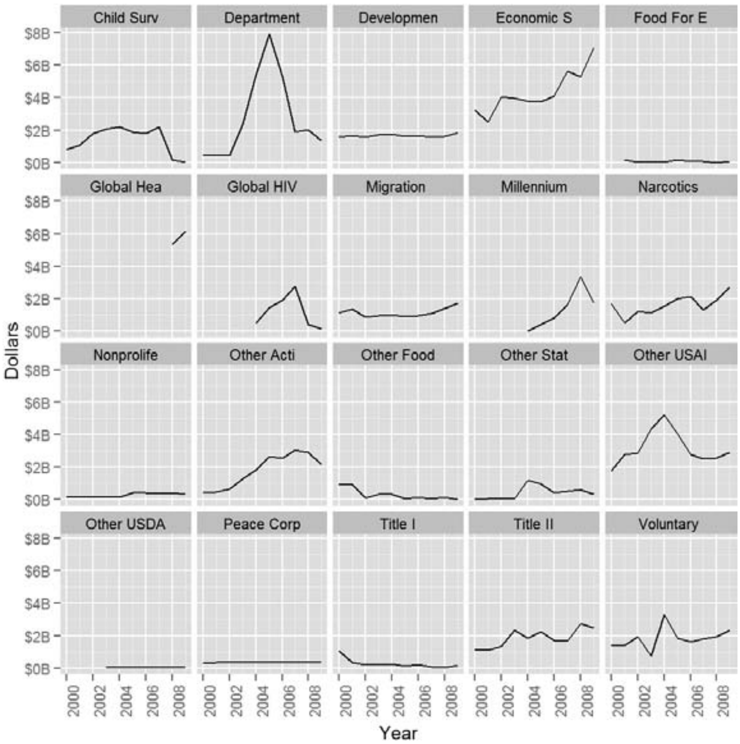

Figure 12.1 Plot of foreign assistance by year for each of the programs.

12.3.2 dcast

Now that we have the foreign aid data melted, we cast it back into the wide format for illustration purposes. The function for this is `dcast`, and it has trickier arguments than `melt`. The first is the data to be used, in our case `melt00`. The second argument is a `formula` where the left side specifies the columns that should remain columns and the right side specifies the columns that should become row names. The third argument is the column (as a character) that holds the values to be populated into the new columns representing the unique values of the right side of the `formula` argument.

```
> cast00 <- dcast(melt00, Country.Name + Program.Name ~ Year,
+     value.var = "Dollars")
> head(cast00)

  Country.Name                              Program.Name 2000
1  Afghanistan                  Child Survival and Health   NA
2  Afghanistan     Department of Defense Security Assistance   NA
3  Afghanistan                     Development Assistance   NA
```

```
4  Afghanistan Economic Support Fund/Security Support Assistance   NA
5  Afghanistan                              Food For Education   NA
6  Afghanistan                    Global Health and Child Survival   NA
     2001     2002      2003       2004       2005       2006
1      NA  2586555  56501189   40215304   39817970   40856382
2      NA  2964313        NA   45635526  151334908  230501318
3 4110478  8762080  54538965  180539337  193598227  212648440
4   61144 31827014 341306822 1025522037 1157530168 1357750249
5      NA       NA   3957312    2610006    3254408     386891
6      NA       NA        NA         NA         NA         NA
        2007       2008       2009
1   72527069   28397435         NA
2  214505892  495539084  552524990
3  173134034  150529862    3675202
4 1266653993 1400237791 1418688520
5         NA         NA         NA
6         NA   63064912    1764252
```

12.4 Conclusion

Getting the data just right to analyze can be a time-consuming part of our work flow, although it is often inescapable. In this chapter we examined combining multiple datasets into one and changing the orientation from column based (wide) to row based (long). We used `plyr`, `reshape2` and `data.table` along with base functions to accomplish this. This chapter combined with Chapter 11 covers most of the basics of data munging with an eye to both convenience and speed.

This page intentionally left blank

General Index

D

H

I

J

K

L

P

Q

R

S

T